Hardware-Driven Nonlinear Activation for Stochastic Computing Based Deep Convolutional Neural Networks

Ji Li[1], Zihao Yuan[1], Zhe Li[2], Caiwen Ding[2], Ao Ren[2], Qinru Qiu[2], Jeffrey Draper[1,3] and Yanzhi Wang[2]

[1]Department of Electrical Engineering, University of Southern California, Los Angeles, CA, USA
[2]College of Engineering and Computer Science, Syracuse University, Syracuse, NY, USA
[3]Information Sciences Institute, University of Southern California, Marina Del Rey, CA, USA
{*jli724,zihaoyua*}*@usc.edu,* {*zli89,cading,aren,qiqiu*}*@syr.edu, draper@isi.edu, ywang393@syr.edu*

***Abstract*—Recently, Deep Convolutional Neural Networks (DCNNs) have made unprecedented progress, achieving the accuracy close to, or even better than human-level perception in various tasks. There is a timely need to map the latest software DCNNs to application-specific hardware, in order to achieve orders of magnitude improvement in performance, energy efficiency and compactness. Stochastic Computing (SC), as a low-cost alternative to the conventional binary computing paradigm, has the potential to enable massively parallel and highly scalable hardware implementation of DCNNs. One major challenge in SC based DCNNs is designing accurate nonlinear activation functions, which have a significant impact on the network-level accuracy but cannot be implemented accurately by existing SC computing blocks. In this paper, we design and optimize SC based neurons, and we propose highly accurate activation designs for the three most frequently used activation functions in software DCNNs, i.e, hyperbolic tangent, logistic, and rectified linear units. Experimental results on LeNet-5 using MNIST dataset demonstrate that compared with a binary ASIC hardware DCNN, the DCNN with the proposed SC neurons can achieve up to 61X, 151X, and 2X improvement in terms of area, power, and energy, respectively, at the cost of small precision degradation. In addition, the SC approach achieves up to 21X and 41X of the area, 41X and 72X of the power, and 198200X and 96443X of the energy, compared with CPU and GPU approaches, respectively, while the error is increased by less than 3.07%. ReLU activation is suggested for future SC based DCNNs considering its superior performance under a small bit stream length.**

***Keywords*—Deep Convolutional Neural Networks; Stochastic Computing; Deep Learning; Activation Function.**

I. Introduction

Deep learning has achieved unprecedented success in solving problems that have resisted the best attempts of the artificial intelligence community for many years [1,2]. Artificial neural networks with deep learning can automatically extract a hierarchy of representations by composing nonlinear modules, which transform the representations at one level (starting from raw input, e.g., pixel values of an image) into representations at a more abstract and more invariant level [2]. Arbitrarily complex functions can be approximated in a sufficiently large neural network using *nonlinear activation functions* [3]. However, in practice the sizes of networks are finite, and the choice of nonlinearity affects both the learning dynamics and the network's expressive power [4].

Recently, the Deep Convolutional Neural Network (DCNN) has achieved breakthroughs in many fundamental applications, such as image/video classification [5,6] and visual/text recognition [7,8]. DCNN is now recognized as the dominant method for almost all recognition and detection tasks and surpasses human performance on certain tasks [2]. Since the deep layered structures require a large amount of computation resources, from a practical standpoint, large-scale DCNNs are mainly implemented in high performance server clusters [9]. The huge power/energy consumptions of software DCNNs prevent their widespread deployment in wearable and Internet of Things (IoT) devices, which emerge with repercussions across the industry spectrum [10]. Hence, there is a timely need to map the latest DCNNs to application-specific hardware, in order to achieve orders of magnitude improvement in performance, energy efficiency and compactness.

Many existing works have explored hardware implementations of DCNNs using GPUs [11] and FPGAs [12,13]. Nevertheless, more efficient design is required to resolve the conflict between the resource-hungry DCNNs and the resource-constrained IoT entities. Stochastic Computing (SC), which uses the probability of 1s in a random bit stream to represent a number, has the potential to enable massively parallel and ultra-low footprint hardware based DCNNs [10,14–17]. In SC, arithmetic operations like multiplication can be performed using simple logic elements and SC provides better soft error resiliency [15,18–20]. In this regard, considerable efforts have been invested in the context of designing Artificial Neural Networks (ANNs), Deep Belief Network (DBNs) and DCNNs using SC components [10,14,16,17,21–23].

One key challenge is designing accurate nonlinear activation function. A small imprecision induced by the convolution and down sampling operations can be significantly amplified by the inaccurate nonlinear activation function, propagated to the subsequent neurons, and further amplified by the activation functions in the following neurons. Hence, without an accurate activation, the network accuracy can easily decrease to an unacceptable level. Besides, while the type of activation has a significant impact on the performance of DCNNs, only two basic Finite State Machine (FSM) based hyperbolic tangent

(tanh) activations are designed for ANNs and DBNs in [10,15], whilst other possible activations have hardly been explored, especially the Rectified Linear Units (ReLUs), which are essential for the state-of-the-art neural networks [24].

In this paper, we first propose three accurate SC-based neuron designs for DCNNs with the popular activation functions that are widely used in the software, i.e., tanh, logistic (or sigmoid), and ReLU. The SC configuration parameters are jointly optimized considering the down sampling operation, block size, and connection patterns in order to yield the maximum precision. Then we conduct a comprehensive comparison of the aforementioned neuron designs using different activations under different input sizes and stochastic bit stream lengths. After constructing the DCNNs using the proposed neurons, we further evaluate and compare the network performance of DCNNs. Experimental results demonstrate that proposed SC based DCNNs have much smaller area and power consumption than the binary ASIC DCNNs, with up to 61X, 151X, 2X improvement in terms of area, power, and energy, respectively, at the cost of $0.0001 - 0.03$ accuracy degradation. Moreover, the SC approach achieves up to 21X and 41X of the area, 41X and 72X of the power, and 198200X and 96443X of the energy, compared with CPU and GPU approaches, respectively, while the test error is increased by less than 3.07%. ReLU activation is suggested for future SC based DCNNs considering its superior accuracy, area, and energy performance under a small bit stream length.

The remainder of the paper is organized as follows. Section II reviews related work. DCNN architecture and related SC components are given in Section III. Section IV presents the three activation function designs in a neuron cell using SC. Section V reports the experimental results, and this paper is concluded in Section VI.

II. Related Work

A. Activation Function Studies

The hyperbolic tangent has generally shown to provide more robust performance than logistic function in DBNs [25]. However, both suffer from the vanishing gradient problem, resulting in a slowed training process or convergence to a poor local minimum [25]. ReLUs do not have this problem, since an activated unit gives a constant gradient of 1. Recently, the parametric ReLU proposed by K. He et al. was reported to surpass human-level performance on the ImageNet large scale visual recognition challenge [24].

B. Hardware-Based DCNN Studies

In order to exploit the parallelism and reduce the area, power, and energy, many existing hardware-based DCNNs have come into existence, including GP-GPUs based DCNNs [26] and FPGA-based DCNNs [12,13]. GP-GPU was the most commonly used platform for accelerating DBNs around 2011 [26]. As for FPGA, M. Motamedi et al. developed an FPGA-based accelerator to meet performance and energy-efficiency constraints of DCNNs [13]. Recently, SC becomes a very attractive candidate for implementing ANNs and DBNs. Y. Ji et al. applied SC to a radial basis function ANN and significantly reduced the required hardware in [21], and hardware-oriented optimization for SC based DCNN was developed in [16]. K. Kim et al. proposed hardware-based DBN using SC components, in which a SC based neuron cell was designed and optimized [10]. A multiplier SC neuron and a structure optimization method were proposed in [14] for DCNN. A. Ren et al. developed a DCNN architecture with weight storage optimization and a novel max pooling design in the SC domain [17]. Besides, Z. Li et al. explored eight different neuron implementations in DCNNs using SC [23].

However, no existing works have explored the ReLU and logistic activations (popular activation choices in software) for hardware based DCNNs using SC.

III. Overview of Hardware-Based DCNN

A. General Architecture of DCNNs

Figure 1 shows a general DCNN architecture that is composed of a stack of convolutional layers, pooling layers, and fully connected layers.

In each convolutional layer, common patterns in local regions of inputs are extracted by convolving a filter over the inputs. The convolution result is stored in a feature map as a measure of how well the filter matches each portion of the inputs. After convolution, a subsampling step is performed by the pooling layer to aggregate statistics of these features, reduce the dimensions of data and mitigate over-fitting issues. A nonlinear activation function is applied to generate the output of the layer. The stack of convolutional and pooling layers is followed by fully connected layers, which further aggregate the local information learned in the convolutional and pooling layers for class discrimination.

By alternating the topologies of convolutional and pooling layers, powerful DCNNs can be built for specific applications, such as LeNet [27], AlexNet [11], and GoogLeNet [28]. With no loss of generality, we use LeNet-5 (i.e., the fifth generation of LeNet for digits recognition) in our discussion and experiments throughout the paper, and the proposed design methodology can accommodate other DCNNs as well.

B. Hardware-Based Neuron Cell

As the basic building block in DCNNs, a neuron performs three fundamental operations, i.e., convolution, pooling, and activation, as shown in Figure 2.

The convolution operation calculates the inner products of input ($x_i's$) and weights ($w_i's$), and the pooling operation

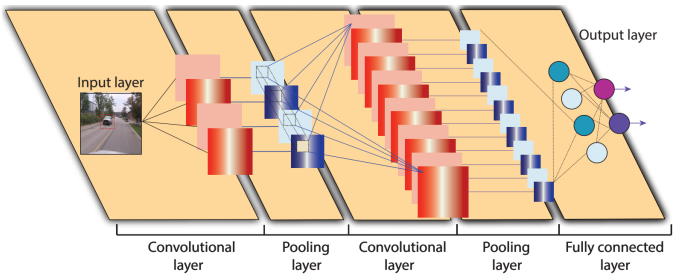

Fig. 1. A general DCNN architecture.

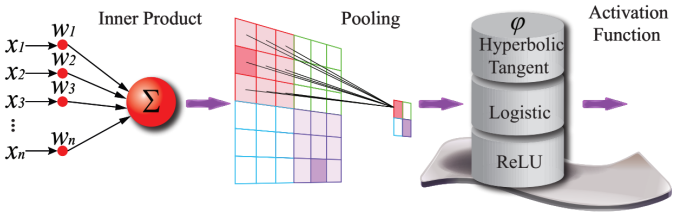

Fig. 2. A hardware-based neuron cell in DCNN.

performs sub-sampling for the inner product results, i.e., generating one result out of several inner products. There are two conventional choices for pooling: max and average. The former chooses the largest element in each pooling region, whereas the latter calculates the arithmetic mean. An activation function is applied before the output, with tanh $f(x) = tanh(x)$, logistic function $f(x) = (1 + e^{-x})^{-1}$, and ReLU $f(x) = max(0, x)$ being popular choices.

From a hardware design perspective, convolution and pooling can be implemented efficiently using basic building blocks (described in Section IV-A) to achieve accurate results. The nonlinear activation function, on the other hand, is usually implemented using Look-Up Tables (LUTs), requiring large memories to achieve adequate precision. Moreover, without careful design and optimization, the nonlinear activation function can result in serious accuracy degradation as it directly affects the final output accuracy of the neuron cell and the imprecision can be amplified by the subsequent calculations and activations. Hence, it is imperative to design and optimize the activation function to achieve sufficient accuracy.

IV. Proposed Hardware-Driven Nonlinear Activation for DCNNs

A. Stochastic Computing for Neuron Design

Stochastic computing (SC) is an encoding scheme that represents a numeric value $x \in [0, 1]$ using a bit stream X, in which the probability of ones is x [15]. For instance, the bit stream "01000" contains a single one in a five-bit stream, thus it represents $x = P(X = 1) = 0.2$. In addition to this unipolar encoding format, another commonly used format is bipolar format, where a numeric value $x \in [-1, 1]$ is processed by $P(X = 1) = \frac{x+1}{2}$. Using bipolar format, the previous example 0.2 could be represented by "10110". Next, we present the fundamental operations of a neuron using SC components.

Convolution. Convolution performs multiplication and addition. An XNOR gate, as shown in Figure 3 (a), is used for multiplication with bipolar stochastic encoding as $c = 2P(C = 1) - 1 = 2(P(A = 1)P(B = 1) + P(A = 0)P(B = 0)) - 1 = (2P(A = 1) - 1)(2P(B = 1) - 1) = a \times b$. Multiplexers (MUXes) can perform bipolar SC addition [15], which randomly selects one input as output. When input size is large, summation using MUXes incurs significant accuracy loss since only one input is selected at a time and all the other inputs are not used. To achieve a good trade-off between accuracy and hardware cost in terms of area, power, and energy, we adopt the Approximate Parallel Counter (APC) developed in [29] for addition instead of MUXes.

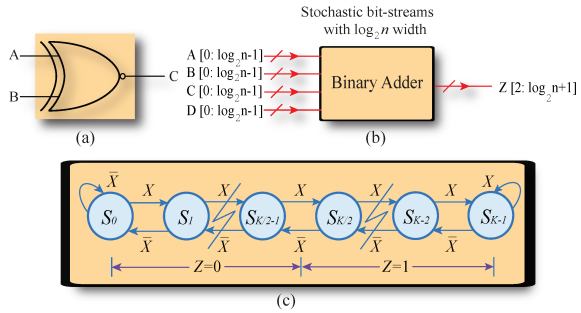

Fig. 3. Stochastic computing for neuron design: (a) XNOR gate for bipolar multiplication, (b) binary adder for average pooling, and (c) FSM-based tanh for stochastic inputs.

Pooling. In the SC domain, average pooling can be implemented efficiently with a simple MUX for stochastic inputs. However, as the outputs of APCs (i.e., inputs of pooling) are binary, MUX cannot be applied here. Instead, we use a binary adder to calculate the sum and remove the last 2 bits of the sum as a division by 4 operation, as illustrated in Figure 3 (b). Note that 4-to-1 average pooling is used in the LeNet-5, whereas max pooling is used in other DCNNs, such as AlexNet [11] and GoogLeNet [28]. The SC-based max pooling design is developed in [17].

Activation. Nonlinear activation is the key operation that enhances the representation capability of a neuron. There are many types of activation functions that have been explored in software DCNNs. Nevertheless, due to the design difficulty, only tanh is designed in the SC domain (e.g., an FSM-based tanh function $Stanh(\cdot)$ proposed in [15], as shown in Figure 3 (c)), and there is a serious lack of studies on other activation functions for SC-based DCNNs. For the convenience of discussions, we use the naming conventions in Table I.

B. Proposed Neuron Design and Nonlinear Activation

We found that directly applying SC to DCNNs leads to severe accuracy degradation which is not acceptable in common cases. The main reason is that the calculation of each computation block (convolution, pooling, and activation) is inaccurate in the SC domain. The overall accuracy of a neuron is affected by number of input streams, bit stream length, hardware configurations, and most importantly the activation functions. In this paper, we design and optimize the neuron structures as well as the activations by jointly considering all the aforementioned factors in order to reach the accuracy level that can be achieved by fixed point binary arithmetic.

To be more specific, we adopt bipolar encoding scheme and average pooling. The proposed neuron designs use XNOR gates and APCs for addition and multiplication (as convolution operation), respectively. Average pooling is implemented using a binary adder (as shown in Figure 3 (b)). The nonlinear activations are designed for binary inputs since the output of APC and the pooling result are binary. The output of the proposed neuron, on the other hand, must be a stochastic bit stream, which will be fed into the neurons in the subsequent layers. The proposed neurons using hyperbolic tangent, stochastic logistic and ReLU activations are as follows:

Algorithm 1: Proposed SC-tanh (ϕ, ψ, q, e)

input : ϕ, ψ are the input bit streams
input : q indicates the q-to-1 average pooling
input : e is internal FSM state number
output: z_k is the k-th stochastic output bit for the SC-tanh neuron

1 $S_{max} \leftarrow e - 1$; /* max state */
2 $S_{bound} \leftarrow e/2$; /* boundary state */
3 $S \leftarrow S_{bound}$; /* current state */
4 **for** $k \leftarrow 1$ **to** m **do**
  /* processing each convolution block */
5 **for** $j \leftarrow 1$ **to** q **do**
    /* inner product calculation */
6 **for** $i \leftarrow 1$ **to** n **do**
7 $p_i^j = x_i^j \odot w_i^j$; /* XNOR multiplication */
8 **end**
9 $t_j = 2 \cdot \sum_{i=0}^{n} p_i^j - n$; /* APC addition */
10 **end**
  /* average pooling and tanh activation */
11 $S \leftarrow S + \frac{\sum_{j=1}^{q} t_j}{q}$
12 **if** $S < 0$ **then** /* saturated counter */
13 $S \leftarrow 0$;
14 **else if** $S > S_{max}$ **then**
15 $S \leftarrow S_{max}$
16 **if** $S > S_{bound}$ **then** /* output logic */
17 $z_k \leftarrow 1$
18 **else**
19 $z_k \leftarrow 0$
20 **end**
21 **end**

SC-tanh: SC-Based Hyperbolic Tangent Neuron. The authors in [10] adopted a saturated up/down counter to implement a binary hyperbolic tangent activation function. Nevertheless, this activation function is designed for DBNs and cannot be applied directly for DCNNs. Algorithm 1 presents the proposed SC-based hyperbolic tangent neuron design *SC-tanh*$(\cdot)$ for DCNNs, where step 5-9 correspond to the inner product calculation using XNOR gates and APCs, step 11 is the average pooling with a binary adder, and step 12-18 are the tanh activation part, which is implemented with a saturated counter. The connections between two layers are based on the filters that execute convolution operations only for portions of the inputs (i.e., receptive fields), which greatly reduce the connections between consecutive layers. The counted binary number generated by APC is taken by the saturated up/down counter which represents the amount of increase and decrease. By setting half of the states' output to 1 and the rest to 0 (i.e., setting boundary state to $S_{bound} = \frac{e}{2}$), this neuron design accurately imitates the hyperbolic tangent function.

TABLE I
NAMING CONVENTIONS IN A STOCHASTIC COMPUTING BASED NEURON

m	the length of bipolar bit stream
q	q-to-1 average pooling
n	input size: the number of input bit streams (or the number of input and weight pairs) in each convolution block
x_i^j	the i-th input bit of the j-th convolution block $(i \in [0, n-1], j \in [0, q-1])$
w_i^j	the i-th weight bit of the j-th convolution block $(i \in [0, n-1], j \in [0, q-1])$
ϕ_j	the set of all the input bits of the j-th convolution block $(\phi_j = \{x_i^j \mid j \in [0, q-1], \forall i \in [0, n-1]\})$
ψ_j	the set of all the weight bits of the j-th convolution block $(\psi_j = \{w_i^j \mid j \in [0, q-1], \forall i \in [0, n-1]\})$
t_j	the sum calculated by the j-th convolution block, which is a $log_2 n$-bit binary number
e	the number of states in the state machine
z_k	the k-th stochastic output bit for the SC-based hyperbolic tangent neuron

SC-logistic: Proposed SC-Based Logistic Neuron. There are two important differences between the hyperbolic tangent neuron and the logistic neuron: (i) unlike the above-mentioned hyperbolic tangent neuron, the output of which is in the range of [-1,1], the logistic neuron always outputs non-negative number (i.e., within [0,1]), and (ii) the y-value of the logistic's midpoint is 0.5 instead of 0 in hyperbolic tangent.

In order to tackle the 1-st difference, we introduce a history shift register array $H[0 : \alpha - 1]$ using α shift registers to record the last α bits of the output stochastic bit streams. A shadow counter is used to calculate the sum of the stochastic bits in the history shift register array, which is denoted by δ, i.e., $\delta = \sum_{i=0}^{\alpha-1} H[i]$. Hence, the proposed SC-logistic keeps tracking the last α output bits and predicts the sign of the current value based on the sum calculated by the shadow counter. To be more specific, if the sum is less than half of the maximum achievable sum $\delta < \frac{\alpha}{2}$, the current value is predicted to be negative. Otherwise, it is predicted as positive (note that value 0 is half 1s and half 0s in bipolar format). As any negative output raises an error, the proposed activation mitigates such errors by outputting 1 (as compensation) whenever the predicted current value is negative.

As for the 2-nd difference, we need to move the y-value of the logistic's midpoint to 0.5. The probability of 1s in a stochastic bit stream X that represents the value $x = 0.5$ is $P(X = 0.5) = \frac{3}{4}$. Hence, we design the logic of activation such that $\frac{1}{4}$ of the states output 0s, whereas the other $\frac{3}{4}$ portion output 1s. This is realized by setting the boundary state to $S_{bound} = \frac{e}{4}$, where e represents the internal FSM state number (in the saturated counter). Note that the boundary value in hyperbolic tangent is $S_{bound} = \frac{e}{2}$ since the midpoint of y-value of tanh is 0 and 0 is represented by a stochastic stream where half of the bits is 0 and the other half is 1 (i.e., $P(X = 0) = \frac{1}{2}$). Algorithm 2 provides the pseudo code of the proposed SC-logistic neuron.

SC-ReLU: Proposed SC-Based ReLU Neuron. According to [2], the ReLU activation $f(x) = max(0, x)$ becomes the most popular activation function in 2015. We apply the same design concept as in the SC-tanh neuron with the following modifications: First, we introduce a history shift register array $H[0 : \alpha-1]$ and its shadow counter δ to predict the sign of the current value, and compensate the negative value in the same way SC-logistic neuron does. Second, unlike the SC-logistic neuron that changes boundary state to $S_{bound} = \frac{e}{4}$ such that the entire waveform is moved up to $y = 0.5$, the SC-ReLU is centered at (0,0) so the boundary state needs to be kept as $S_{bound} = \frac{e}{2}$. Noticing the similarities between the SC-logistic and SC-ReLU, we introduce a configuration bit β and combine these two neurons, and Algorithm 2 provides the pseudo code of the proposed SC-logistic and SC-ReLU neuron.

Note that in the proposed neurons, the FSM state number e is generated using a simple binary search algorithm under different input sizes to yield the highest precision. Unlike the software implementation that has limited allowable degree of parallelism and high coordination overheads, the proposed specific hardware based neurons can execute the commands in

Algorithm 2: Proposed SC-logistic/ReLU ($\phi, \psi, \alpha, \beta, q, e$)

input : ϕ, ψ are the input bit streams
input : α is the number of registers in the temporary array
input : β is the configuration bit. 1:SC-logistic, 0:SC-ReLU
input : q indicates the q-to-1 average pooling
input : e is internal FSM state number
output: z_k is the k-th stochastic output bit for the SC-logistic/ReLU neuron

$S_{max} \leftarrow e - 1$; /* max state */
if $\beta == 1$ **then** /* boundary state configuration */
  $S_{bound} \leftarrow e/4$; /* SC-logistic */
else
  $S_{bound} \leftarrow e/2$; /* SC-ReLU */
end
$S \leftarrow S_{bound}$; /* current state */
$r \leftarrow \alpha - 1$; /* r is an iterator */
$H[0 : \alpha - 1] \leftarrow 0$; /* initialize history array */
$\delta \leftarrow 0$; /* initialize shadow counter */
for $k \leftarrow 1$ **to** m **do**
  if $\delta < \frac{\alpha}{2}$ **then**
    $z_k \leftarrow 1$; /* negative value compensation */
  else
    for $j \leftarrow 1$ **to** q **do**
      for $i \leftarrow 1$ **to** n **do**
        $p_i^j = x_i^j \odot w_i^j$; /* XNOR multiplication */
      end
      $t_j = 2 \cdot \sum_{i=0}^{n} p_i^j - n$; /* APC addition */
    end
    $S \leftarrow S + \frac{\sum_{j=1}^{q} t_j}{q}$
    if $S < 0$ **then** /* saturated counter */
      $S \leftarrow 0$
    else if $S > S_{max}$ **then**
      $S \leftarrow S_{max}$
    end
    if $S > S_{bound}$ **then** /* output logic */
      $z_k \leftarrow 1$
    else
      $z_k \leftarrow 0$
    end
  end
  while $r \geq 1$ **do**
    $H[r] \leftarrow H[r-1]$; /* update the history array */
    $r \leftarrow r - 1$
  end
  $H[0] \leftarrow z_k$
  $\delta \leftarrow \sum_{r=0}^{\alpha-1} H[r]$; /* update the shadow counter */
  $r \leftarrow \alpha - 1$
end

Algorithm 1 and 2 fully parallelly. Figure 4 (a), (b), and (c) show that the proposed SC-tanh, SC-logistic, and SC-ReLU neurons and their corresponding software results are almost identical to each other. Detailed error analysis is provided in Section V.

V. Experimental Results

We now present (i) performance evaluation of the proposed SC neurons, (ii) comparison with binary ASIC neurons, and (iii) DCNN performance evaluation and comparison. The neurons and DCNNs are synthesized in Synopsys Design Compiler with the 45nm Nangate Library [30] using Verilog.

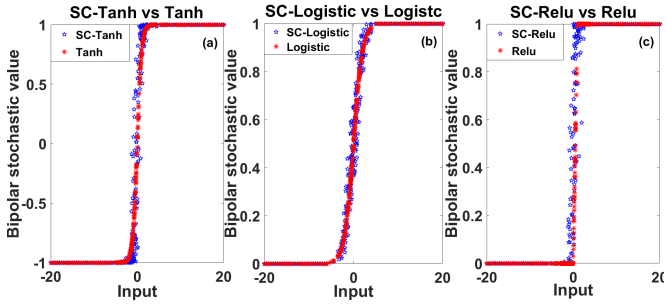

Fig. 4. The result comparison between the proposed SC neuron (bit stream $m = 1024$) and the corresponding original software neuron: (a) SC-tanh vs Tanh, (b) SC-logistic vs Logistic, and (c) SC-ReLU vs ReLU.

A. Performance Evaluation and Comparison among the Proposed Neuron Designs

For each neuron, the accuracy is dependent on (i) the bit stream length m, and (ii) the input size n. Longer bit stream length yields higher accuracy, and the precision can be leveraged by adjusting the bit stream length without hardware modification. On the other hand, the input size n, which is determined by the DCNN topology, affects both the accuracy of the neuron as well as the hardware footprint.

Thus, considering the aforementioned factors, we evaluate the inaccuracy of the proposed SC-tanh, SC-logistic and SC-ReLU neurons under a wide range of bit stream lengths and input sizes, as shown in Figure 5 (a), (b) and (c), respectively. The corresponding hardware costs of the proposed SC-tanh, SC-logistic and SC-ReLU neurons are shown in Figure 6 (a), (b) and (c), respectively. One can observe that the precisions of SC-logistic and SC-ReLU neurons consistently outperform SC-tanh neurons under different combinations of input size and bit stream length, whereas the area, power and energy of SC-tanh neurons are slightly lower than the other two neurons. Moreover, SC-logistic and SC-ReLU neurons have better scalability in terms of accuracy than the SC-tanh neuron. Note that the inaccuracy here is calculated by comparing with the software neuron results. A lower inaccuracy only indicates that the hardware neuron with this type of activation is closer to its software version, but this does not indicate that a DCNN implemented with this neuron can yield a lower test error.

B. Comparison with Binary ASIC Neurons

We further compare the proposed SC-based neurons with the binary ASIC hardware neurons. The input size is set to 25 since most neurons in LeNet-5 DCNN are connected to 5×5 receptive fields. The binary nonlinear activations logistic and ReLU are implemented using LUTs, whereas binary ReLU is built with a comparator and a MUX. Clearly, the number of bits in fixed-point numbers affects both the hardware cost and the accuracy. To make the comparison fair, we use the minimum fixed point (8 bit) that yields a DCNN network accuracy that is almost identical to the software DCNN (with < 0.0003 difference in network test error). Table II shows the neuron cell performance comparison between the proposed SC neurons and the 8 bit binary ASIC neurons. Compared with binary ASIC neurons, the proposed SC neurons achieve up to

TABLE II
Neuron Cell Performance Comparison with 8 Bit Fixed Point Binary Implementation when $n = 25$ and $m = 1024$

	8 bit binary ASIC			proposed SC neuron			improvement		
	tanh	logistic	ReLU	tanh	logistic	ReLU	tanh	logistic	ReLU
power (μW)	34760	34760	32176	173	223	231	201X	156X	139X
energy (fJ)	128267	128267	94346	858	1130	1147	149X	114X	82X
area (μm^2)	42319	42319	35506	699	916	916	61X	46X	39X

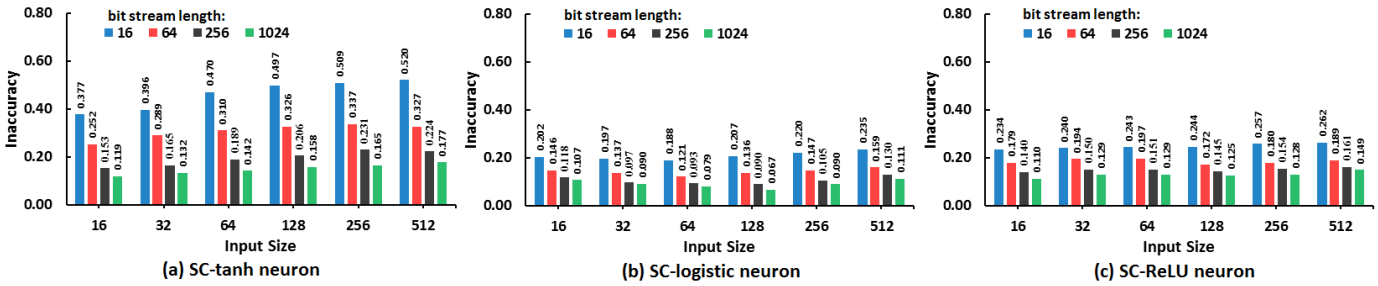

Fig. 5. Input size versus absolute inaccuracy under different bit stream lengths for (a) SC-tanh neuron, (b) SC-logistic neuron, and (c) SC-ReLU neuron.

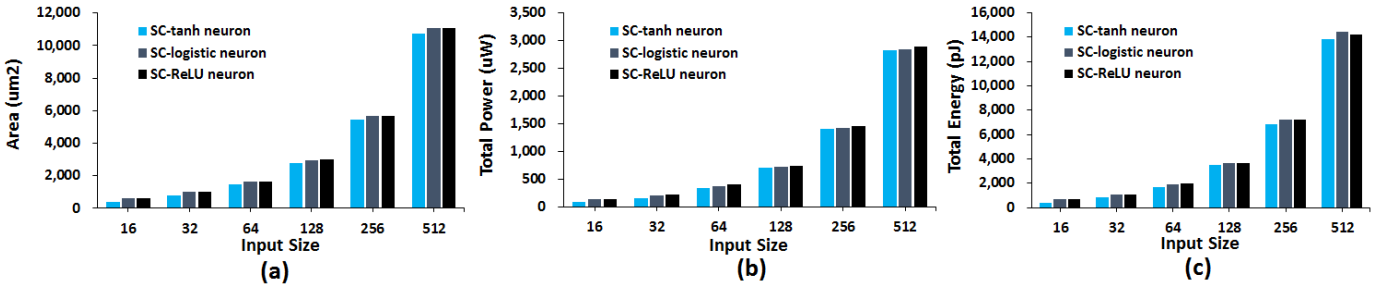

Fig. 6. Input size versus (a) area, (b) total power, and (c) total energy for the neuron designs using tanh, logistic and ReLU activation functions.

TABLE III
COMPARISON AMONG SOFTWARE DCNN, BINARY ASIC DCNN, AND VARIOUS SC BASED DCNN DESIGNS IMPLEMENTING LENET 5

activation	approach	bit stream	valid. error	test error	area (mm2)	power (W)	energy (μJ)
tanh	SC	1024	1.74%	1.41%	12.5	3.1	15.8
		512	1.57%	1.65%			7.9
		256	1.71%	1.61%			3.9
		128	1.84%	2.13%			2.0
		64	2.37%	2.34%			1.0
	binary	-	1.42%	1.34%	769.3	470.0	2.0
	CPU	-	1.41%	1.34%	263	130.0	198200
	GPU	-	1.41%	1.34%	520	225.0	96443
logistic	SC	1024	3.98%	4.49%	15.8	3.9	20.1
		512	4.35%	4.70%			10.0
		256	4.24%	4.34%			5.0
		128	5.23%	5.58%			2.5
		64	6.30%	6.06%			1.3
	binary	-	2.88%	3.01%	769.3	585.7	2.4
	CPU	-	2.87%	2.99%	263	130.0	198200
	GPU	-	2.87%	2.99%	520	225.0	96443
ReLU	SC	1024	1.69%	1.69%	15.8	3.9	20.3
		512	1.67%	1.69%			10.1
		256	1.67%	1.63%			5.1
		128	1.65%	1.67%			2.5
		64	1.67%	1.63%			1.3
	binary	-	1.65%	1.65%	664.9	557.5	1.8
	CPU	-	1.64%	1.64%	263	130.0	198200
	GPU	-	1.64%	1.64%	520	225.0	96443

201X, 149X, and 61X improvement in terms of power, energy, and area, respectively, indicating significant hardware savings.

C. DCNN Performance Evaluation and Comparison

To evaluate the network performance, we construct the LeNet-5 DCNNs using the proposed SC neurons as well as the 8 bit binary neurons in a pipelined manner. LeNet 5 is a widely-used DCNN structure with a configuration of 784-11520-2880-3200-800-500-10. The DCNNs are evaluated with the MNIST handwritten digit image dataset [31], which consists of 60,000 training data and 10,000 testing data. We apply the same training time in software so as to make a fair comparison among different activations.

Table III concludes the performance of DCNNs using CPU, GPU, binary neurons and the proposed SC neurons. The CPU approach uses two Intel Xeon W5580, whereas the GPU approach utilizes NVIDIA Tesla C2075. Note that the power for software is estimated using Thermal Design Power (TDP), and the energy is calculated by multiplying the run time and TDP. On the other hand, the power, energy, and area for hardware are calculated using the synthesized netlists with Synopsys Design Compiler. One can observe that for each type of activation, the proposed SC based DCNNs have much smaller

area and power consumption than the corresponding binary DCNNs, with up to 61X, 151X, 2X improvement in terms of area, power, and energy, respectively. Note that though the the binary ASIC has a competitive energy performance, it is an ideal pipelined structure. The extremely large area ($> 600mm^2$) and power ($> 400W$) makes binary ASIC unpractical for implementation. Moreover, Table III shows that the proposed SC approach achieves up to 21X and 41X of the area, 41X and 72X of the power, and 198200X and 96443X of the energy, compared with CPU and GPU approaches, respectively, while the error is increased by less than 3.07%.

Among different activations, with a long bit stream ($m > 128$), SC-tanh is the most accurate. Otherwise ($m \leq 128$), SC-ReLU has the highest precision. SC-logistic has the lowest precision due to the following reasons: (i) logistic activation in software has the worst accuracy performance, and (ii) the imprecision of activation (if larger than a certain threshold) amplifies the inaccuracy in DCNNs. The bit stream length can be reduced to improve energy performance. One important observation is that the proposed SC-ReLU has better scalability than SC-tanh (i.e., with bit stream length decreasing, the accuracy degradation of SC-ReLU is slower than SC-tanh). Hence, ReLU activation is suggested for future SC based DCNNs considering its superior accuracy, area, and energy performance under a small bit stream length (e.g., $m = 64$). Note that the small bit stream length leads to significant improvement in terms of delay and energy performance.

VI. Conclusion

In this paper we presented three novel SC neurons designs using tanh, logistic, and ReLU nonlinear activations. LeNet-5 DCNNs were constructed using the proposed neurons. Experimental results on the MNIST dataset demonstrated that compared to the binary ASIC DCNN, the proposed SC based DCNNs were able to significantly reduce the area, power and energy footprint with a small accuracy degradation. ReLU was suggested for future SC based DCNNs implementations.

References

[1] Y. Bengio, "Learning deep architectures for ai," *Foundations and trends® in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.

[2] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[3] Y. Cho and L. K. Saul, "Large-margin classification in infinite neural networks," *Neural computation*, vol. 22, no. 10, pp. 2678–2697, 2010.

[4] F. Agostinelli, M. Hoffman, P. Sadowski, and P. Baldi, "Learning activation functions to improve deep neural networks," *arXiv preprint arXiv:1412.6830*, 2014.

[5] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[6] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, "Large-scale video classification with convolutional neural networks," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2014, pp. 1725–1732.

[7] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell, "Decaf: A deep convolutional activation feature for generic visual recognition." in *ICML*, 2014, pp. 647–655.

[8] Y. Li, Z. Li, and Q. Qiu, "Assisting fuzzy offline handwriting recognition using recurrent belief propagation," in *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, Dec 2016, pp. 1–8.

[9] A. Ardakani, F. Leduc-Primeau, N. Onizawa, T. Hanyu, and W. J. Gross, "Vlsi implementation of deep neural network using integral stochastic computing," *arXiv preprint arXiv:1509.08972*, 2015.

[10] K. Kim, J. Kim, J. Yu, J. Seo, J. Lee, and K. Choi, "Dynamic energy-accuracy trade-off using stochastic computing in deep neural networks," in *Proceedings of the 53rd Annual Design Automation Conference*. ACM, 2016, p. 124.

[11] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[12] A. Rahman, J. Lee, and K. Choi, "Efficient fpga acceleration of convolutional neural networks using logical-3d compute array," in *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2016, pp. 1393–1398.

[13] M. Motamedi, P. Gysel, V. Akella, and S. Ghiasi, "Design space exploration of fpga-based deep convolutional neural networks," in *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2016, pp. 575–580.

[14] J. Li, A. Ren, Z. Li, C. Ding, B. Yuan, Q. Qiu, and Y. Wang, "Towards acceleration of deep convolutional neural networks using stochastic computing," in *The 22nd Asia and South Pacific Design Automation Conference (ASP-DAC). IEEE*, 2017.

[15] B. D. Brown and H. C. Card, "Stochastic neural computation. i. computational elements," *IEEE Transactions on computers*, vol. 50, no. 9, pp. 891–905, 2001.

[16] Z. Li, A. Ren, J. Li, Q. Qiu, Y. Wang, and B. Yuan, "Dscnn: Hardware-oriented optimization for stochastic computing based deep convolutional neural networks," in *Computer Design (ICCD), 2016 IEEE 34th International Conference on*. IEEE, 2016, pp. 678–681.

[17] A. Ren, J. Li, Z. Li, C. Ding, X. Qian, Q. Qiu, B. Yuan, and Y. Wang, "Sc-dcnn: highly-scalable deep convolutional neural network using stochastic computing," *arXiv preprint arXiv:1611.05939*.

[18] J. Li and J. Draper, "Joint soft-error-rate (ser) estimation for combinational logic and sequential elements," in *VLSI (ISVLSI), 2016 IEEE Computer Society Annual Symposium on*. IEEE, 2016, pp. 737–742.

[19] A. Alaghi and J. P. Hayes, "Survey of stochastic computing," *ACM Transactions on Embedded computing systems (TECS)*, vol. 12, no. 2s.

[20] J. Li and J. Draper, "Accelerating soft-error-rate (ser) estimation in the presence of single event transients," in *Proceedings of the 53rd Annual Design Automation Conference*. ACM, 2016, p. 55.

[21] Y. Ji, F. Ran, C. Ma, and D. J. Lilja, "A hardware implementation of a radial basis function neural network using stochastic logic," in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*. EDA Consortium, 2015, pp. 880–883.

[22] K. Sanni, G. Garreau, J. L. Molin, and A. G. Andreou, "Fpga implementation of a deep belief network architecture for character recognition using stochastic computation," in *Information Sciences and Systems (CISS), 2015 49th Annual Conference on*. IEEE, 2015, pp. 1–5.

[23] Z. Li, A. Ren, J. Li, Q. Qiu, B. Yuan, J. Draper, and Y. Wang, "Structural design optimization for deep convolutional neural networks using stochastic computing," 2017.

[24] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 1026–1034.

[25] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. ICML*, vol. 30.

[26] D. C. Ciresan, U. Meier, J. Masci, L. Maria Gambardella, and J. Schmidhuber, "Flexible, high performance convolutional neural networks for image classification," in *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, vol. 22, no. 1, 2011, p. 1237.

[27] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[28] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1–9.

[29] K. , J. Lee, and K. Choi, "Approximate de-randomizer for stochastic circuits," *Proc. ISOCC*, 2015.

[30] Nangate 45nm Open Library, Nangate Inc., 2009. [Online]. Available: http://www.nangate.com/

[31] L. Deng, "The mnist database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine*, vol. 29, no. 6.